A Wire Oriented Mask Geometry Editor

By

Stephen Trimberger

Technical Report #2870

June 1, 1979

Computer Science Department

California Institute of Technology

Pasadena, California 91125

Silicon Structures Project

sponsored by

Burroughs Corporation, Digital Equipment Corporation,

Hewlett-Packard Company, Honeywell Incorporated,

International Business Machines Corporation,

Intel Corporation, Xerox Corporation,

and the National Science Foundation

A Wire Oriented Mask Geometry Editor

Table of Contents

1.0 Introduction

Traditionally, integrated circuit design has been done at the mask geometry level. Recently, however, much effort has been put into higher-level design systems for generating mask data, such as the stick diagram systems at Hewlett-Packard [1] and at Caltech [2] and the Bristle Blocks system [3] at Caltech. These systems cause one to re-examine the purpose of a mask geometry editor.

Two stages of integrated circuit production at Caltech are currently done at the mask geometry level: chip detailing, the actual physical description of the chip; and chip integration, the assembly of many pieces of the chip into a unit. Higher level systems are attacking these areas of design, but those higher level systems are still experimental. None have produced chips other than their small test cases. In addition, the high level systems are often very process dependent. A stick diagramming system is made to generate mask geometry for a specific process and can be very difficult to modify to work for another process.

Bristle Blocks takes as input, low level cells or pieces of a chip which are already defined as mask geometry. It seems incongruous that a high level system like Bristle Blocks requires its users to define their cells with colored pencils on graph paper. Bristle Blocks users could very definitely benefit from a mask geometry editor.

A pure mask geometry editor with no built in design rules does not suffer from process-dependence. It can be used unaltered for NMOS, CMOS, I^2L or PC board layout. A mask geometry editor enables designers to optimize specific cells quickly. Although dense cells are not imperative for the application for which Geom, the Caltech mask geometry editor is expected to be used, custom LSI for computer system designers, dense cells are often important to justify the existence of a chip. If the chip is too big, it simply cannot be made.

My contention is not that mask geometry editing is better or preferable to higher level design aids, but simply that the mask geometry editor is a more versatile tool and may be used in some situations where the others cannot produce integrated circuits. Geom complements other design systems for the capture and manipulation of physical integrated circuit data.

This thesis describes an integrated circuit mask geometry editor called Geom. Geom

can be used either as a target representation for other representations in the Desktop system [6], or as a stand-alone integrated circuit graphics editor.

2.0 Objectives

The main objectives of Geom are:

- provide a tool which can be used by the entire integrated circuit design community at Caltech.

- provide users of that tool with a small set of commands which are easy to learn, powerful, and interactive in nature.

- allow hierarchical design by enabling easy use of instances of cells

- allow users to manipulate a limited set of graphic primitives, specifically wires, in order to reduce the size of the user interface, to provide fewer and more easily learned commands, and to give users a graphic primitive which is appropriate for circuit design.

- allow creation of a wider range of graphic primitives including boxes and polygons directly in the database by other representations in the Desktop system, and provide limited editing of these graphic features through the user interface. Thus we easily gain compatibility with existing systems while maintaining the simplicity of our own.

- place restrictions on the input data in the interest of providing the designer with a restricted, yet powerful graphic tool for integrated circuit design. These restrictions include coordinate gridding and limiting wire segments to horizontal, vertical and diagonal orientations. These restrictions need not apply to data generated from other representations.

- investigate higher-level graphic primitives, wires, to ease the designer's burden without losing process independence.

- provide a base from which to investigate the combination of graphical and procedural

forms of mask geometry.

3.0 Environment

3.1 Hardware Environment

Geom is written in Simula and runs on the DEC-20. Graphic output is provided by a raster scan color display, called the "Charles Terminal". Graphic input is provided by the Xerox mouse and keyset (figure 1). Hardcopy can be gotten from a Hewlett-Packard 7221A, a four color line drawing plotter. The Charles terminal and the interface to it and the graphic input devices are described in [4] and [5]. The plotter is treated in [17].

3.2 Software Environment

Geom is built on Desktop (figure 2), Jim Rowson's editor building system which supports multiple windows and multiple data representations [6]. Desktop supports a hierarchical design methodology and provides facilities for organizing and selecting cells in the hierarchy for viewing and editing. Desktop allows the user to enter editors for manipulating the various representations of the data. Geom is one of these editors.

Desktop provides system-level support for the design such as facilities for the creation and management of viewing windows. This system-level support need not be replicated in each of the editors. Desktop provides a vehicle which can be used to tie together representations, to facilitate consistency checking or generation of one data representation from another. No editor is constrained by the Desktop system in its operation or its data, with the exception that it must interface to the Desktop system program and data in a pre-determined fashion. Restrictions on the data may imposed later to allow checking of consistency among the data representations.

All viewing windows in Desktop contain a cell being viewed and a representation which provides viewing and editing features. Therefore, every element created in an editor is inside a cell. There can be many windows on the screen simultaneously showing different views of the same cell or different cells, in the same representation or in

different representations. Only one window can be active at a time, meaning that the editor in that window will get control from Desktop. Figure 3 shows the structure of Desktop's code and data.

Desktop has a "browser", a special window for organizing the names of cells. The user may create cells with the browser, and use the browser to categorize the names of the cells. Therefore, there are no facilities in Geom for the creation and organization of cells. Desktop handles all window creation, destruction and placement, so Geom ignores those problems also. Desktop contains commands for reading and writing data to a permanent database file and calls Geom to read and write its data piece.

4.0 Cells, Instances and the Design Hierarchy

The design is a hierarchy of cells (figure 4). A cell may contain any number of primitives, wires, and instances of other cells.

An instance correspond to a macro call on the another cell. The instance represents the data of the other cell transformed by the transformation matrix in the instance and placed into the containing cell. The instance must be manipulated as a unit. In order to edit the data in the instance, one must edit the cell which defines the instance. Instances may be displayed on the screen as either the interface specification, their bounding box and connections; or as the transformed image of their entire contents.

Geom relies on the Desktop facilities for defining and organizing cells. Once inside a Desktop window, the user may modify the cell's geometry, but it is not possible to create new cells in Geom. The Geom piece of the cell specification has parameters of the cell particular to the geometry representation, specifically, the bounding box, the connectors, the mask data contents of the cell and the version number. The version number is used to make update checks.

It is possible to recursively reference cells, but this should be avoided, as there is no meaningful way to interpret recursive cell references. All data transformations can be applied to the instance. These do not affect the defining cell at all. If the defining cell is modified, all instances are automatically updated without warning. This can

cause problems if the interface changes.

5.0 Wires

In order to reduce the number of commands and simplify the user interface, only one graphic primitive, the wire, is provided for the user. Different systems have chosen different single primitives. General graphic systems are built around polygons as graphic primitives. However, we are dealing with integrated circuits, not arbitrary shapes in two dimensions. We do not need the generality offered by polygons.

Icarus [7] uses only rectangles. The rectangles are specified by marking the center path segment of the rectangle. A default width is supplied automatically. The user can imagine drawing a conducting path, which is made of many rectangles. Unfortunately, when the user tries to manipulate the path as a unit, he finds that it is not a wire, merely a collection of rectangles. The wire shatters into its component pieces.

Stick diagramming systems under development [1, 2] have a wire as the major primitive. The wire expresses the connectivity of the design, allowing the user to quickly input his sticks. A user of a mask geometry editor can also benefit from this feature of wires.

Geom uses a wire as its only graphic primitive. A wire consists of a width, a layer and a centerpath. To create a wire in Geom, the user specifies the path, giving the points of the center path. The default width and layer are supplied by the system state. He can then manipulate the wire as a single item, not as a collection of rectangles as is the case in Icarus. Geom provides commands for modifying wire widths and paths, and the resultant commands provide some very helpful features. These will be discussed in the section "Wire Edits".

Wires provide a good model for integrated circuit description, that of a conducting path. This model is not quite as intuitive when dealing with depletion mode implant, for example, but it is not a major stumbling block when applying implant. The wire provides a good model most of the time, and provides an adequate model otherwise.

If the default width is set to an appropriate value, the designer need not continually concern himself with checking the widths of features to ensure they meet design rule specifications. Just as mental design rule checking is simplified by the path width, mechanical DRC can be made easier. Minimum feature dimensions are trivial to check, minimum separation can be done with a simple check on the centerpaths. There are, of course, many cases that are not improved by the wire model, such as spacing between active transistor area and contact cuts, but these checks are not made more difficult by the user of wires.

Because the wire represents a conducting path, the wires in a cell exhibit the connectivity of the cell, giving a first cut at a netlist. More importantly, during the editing session, designers can carry out operations on pieces of the design while the editor guarantees that the connectivity is preserved. Wires aid wiring analysis in a way boxes and polygons cannot because boxes and polygons do not exhibit connectivity.

Preparation time for circuit simulation can be reduced by use of wires as graphic primitives. The length of a wire is the length of the centerpath. The area of a wire is the width times the length.

In some cases, wires are simply not the feature desired. Overglass cuts, for example, cannot be considered conducting paths by any stretch of the imagination. The user would like to manipulate features on the overglass layer as boxes. A two-point wire is very similar to a box, and rather than provide another graphic primitive for the overglass layer, facilities are provided in Geom for stretching lengths and widths of wires. These facilities correspond closely to the operations one can do on a box.

Sometimes the designer must create a polygon. There are four things to do in Geom when a polygonal feature is needed:

1) Make a convoluted wire which has the desired shape. This destroys the connectivity and has very little intuitive value.

2) Make up the polygon of many wires. This might preserve the connectivity, depending on the set of wires chosen to represent the polygon. The polygon thus created could be put into a cell, the instances of which can be manipulated as a unit,

while the wires that make up the cell can be manipulated to alter the shape of the "polygon".

3) Enter another representation in the Desktop system which will create polygons for you in the Geom database. The Geom database does have boxes and polygons, but the Geom user interface has no facilities for creating and modifying them. Other representations, however, are able to create boxes and polygons in the Geom data, and Geom can Delete, Copy, Move, Mirror and Rotate them.

4) Give up on polygons. Many uses of polygons can be avoided by simply envisioning the feature to be created as a path with a width rather than as an area to be surrounded.

All in all, polygonal features are difficult to make in Geom. This is not considered a serious drawback as the Geometry Editor was made to attack the problem of fast custom integrated circuit design. Optimum density was not considered a primary design goal. Polygons can always be avoided by devoting more space to the cell in question. The speed of the design process and simplicity of use of the system are enhanced by the use of a single graphic primitive, in particular, the wire. Retaining these advantages outweighs the marginal value of permitting micron-hacking (or sub-micron hacking in VLSI).

6.0 Types of Wires

Figure 5 shows three types of wire ends. Each has its advantages and disadvantages.

The first type of wire end has the end of the wire at exactly the endpoint. This is especially well suited to interactive graphic applications where the user may point at the spot where he wants the wire to end. However, when two wires share an endpoint, they do not merge in a nice way. There is a gap missing in the connection. In addition, it is possible to make wires that are shorter than the default width. If the default width is the design rule minimum width, then a design rule may be inadvertantly violated.

The second wire end type has the wire extend in a semicircle past the endpoint. This definition of the wire has an especially clean mathematical description. The wire is the locus of points within half the width of the centerpath. This is the way CIF [8] wires are defined. Design rule checking of wire separation is especially easy with this type of wire. Two wires sharing an endpoint merge easily. However, no pattern generator can make a circle, and circles are hard to draw quickly in an interactive graphic system. In both cases, circles are usually approximated by octagons. So the clean definition of the wire does not accurately express the implementation.

The third type of wire end has the wire extend half the width past the endpoint. This is the usual compromise to semicircular ends. These wires are easier to draw and to convert for pattern generation. However, wires sharing a point sometimes leave overhanging "ears" which might destroy the electrical properties of an analog circuit or simply get in the way in any design. This third type of wire is the one used in Geom. A wire consisting of two coincident points creates a square. It is impossible to create a feature with any dimension less than the width of the wire. If the default widthof a wire is the design rule minimum width, it is impossible to create a feature which violates the minimum width design rule.

7.0 Wire Edits

In an interactive system, one must be able to modify existing features on the display quickly and easily. When dealing with rectangles, the operations to provide are rather limited: stretch the rectangle in either dimension. The major operations on polygons are insert, delete and move vertices. Wires have properties of both boxes and polygons, there is a width, which is like a box dimension, and a path, which is like a polygon boundary. Both can be modified. The goal in Geom was to allow modification of the parameters of a wire without implementing all the box manipulaion commands and all the polygon manipulation commands. Geom tries to treat wires as wires during editing, providing the user with commands which operate on his mental model of the feature.

The implementation of the stretch operation in a wire based system requires two kinds of operations, changing the length of segments of the wire and changing the width of the wire. In Geom, these are implemented as two completely different operations, so

the user must think about how he is modifying a wire before actually doing it. This might seem like a problem, but only if one's model of the features is a box. The heterogeneous method provides a much more powerful editing capability when dealing with wire objects. And, in most edits, if the object being edited is visualized as a wire, the user would expect one command for each kind of edit.

One stretch command changes the width of a wire. The nearer edge of the wire is moved to the cursor location, the other edge is unmoved. The width of the wire and the center path are changed in order to achieve this box-stretching effect. This gives more than simple box stretching when used on a real wire. Enlarging a power bus is simple with the stretch width command. The far edge does not move, so one need not worry about accidentally violating some design rule between the far edge of the wire and some other feature in the cell. The whole wire becomes wider, no matter how many bends it has, and the connectivity of the power bus is unchanged.

Geom has a command to break a wire at a selected point. Each piece can then be manipulated as an independent element. Breaking a wire, then deleting one piece, is equivalent to deleting a set of points on the wire. A command to delete points from a wire probably would not be used very often, so it would tend to clutter the user interface. The break command has wider applicability.

Conversely, Geom has a command to weld together two wires. The wires must be on the same layer, must have the same width and must share a point. The merged pieces then make up a single wire which can be manipulated as a unit. Once the two wires are welded, the designer need no longer worry about accidentally destroying the connectivity of that connection, the wire is a unit. The weld command also removes overhanging "ears" at the connection point which occur because Geom's wires extend half the width past the endpoint. This problem is discussed in the section titled "Types of Wires".

During the editing session, the user creates a wire by giving a command to add wire segments. The segments are added to the current wire. A wire stops being current when the user stops giving commands to create segments and change the display characteristics. If the user creates segments after an intervening command, he will be creating a new wire, not extending the old one. A command is included to extend an existing wire by making it current. The effect of extending is the same as creating

a new wire then welding the two. Since extending a wire makes up a large part of the use of welding, and since the use of long wires is encouraged in order to preserve the connectivity of the cell, there is a command for this operation.

The most important wire editing command is the command to move selected points on selected wires. If all the points on a wire are selected, the operation is equivalentto a move for that wire. If not all of the points are selected, then the operation is a stretch along a wire segment. Geom ensures that all segments of the wires with moved points will still be vertical, horizontal or diagonal. The real power of this command is that the user can pick up a piece of the design, all the points in one section of the cell, not just all the elements in one section, and move the piece of the design as a unit. The piece will move where appropriate and stretch where appropriate. Because we are dealing with wires, the connectivity is preserved. In a non-wire-based system, this operation would have to be carried out in two steps, a move and a stretch, increasing the probability of error.

8.0 Input/Output

Geom produces CIF2.0 [8] output, although perhaps it should not. CIF2.0 is meant to be an intermediate form for producing pattern generator output. It is improper for all our programs to produce pattern generator output. Instead, we should output to files in an agreed-upon database format, and have one program translate from that database to CIF.

Such a database exists, it is built on a relational database [12]. Relations for the database for our application have been proposed [11]. The input output commands are in Desktop, because each cell consists of pieces from many representations. Desktop passes control to each of the representations for them to output their data in their own relations in the database. Representations must refrain from using each others relation names, but there is no restriction on the number of relations a particular representation may have nor is there a restriction of the data in those relations. However, restrictions on the data in the tuples must be made so that programs that do consistency and validity checking can function properly.

Geom's data is separated into seven relations. All relations have the cell name as the

first entry. This is used as a search key to extract all the data in a given cell. First there is the cell-wide data in relation Geometry. This is merely the cell name followed by the number of connectors and the user's bounding box of the cell. Each connector is kept as one tuple in the relation Connectors. Each connector has the cell name, the connector name, the box which represents the connector and the layer for that connector. These two relations give all the data necessary to show the cell "invisible", the outline and the connectors.

The Instances relation consists of the cell name, instance name and the six values that make up the transformation matrix [15]. This relation is identical to one that is used for stick diagram instances, but the two must be separate relations because the data in the tuples is different.

Wires consist of the cell name, the node name, the wire name, width and layer. Points on the wire are stored in a separate relation Points, which has cell name, wirename, point number and the X,Y coordinates of the point. The node name field is not usedby the geometry editor as it is intended to be used to tie together the physical and logical data representations of the cell data. Geom has no facilities to do this, buta post-processor or another editor in the Desktop system might provide that information.

The other two graphic primitives are handled in the obvious fashion. Rectangles are made of a cell name, node name, the coordinates of two diagonal corners and a layer. Polygons have the cell name, the node name, polygon name and layer. Polygon names and wire names are mutually exclusive, so the polygon uses the Points relation to store the vertices of the polygon.

Input of a database file into the Desktop system proceeds as follows: Desktop receives the command from the user and reads its data from the file. The desktop data consists of cell categories and cell names. Then, for each cell, Desktop creates the in-store cell entry and calls each representation to read in its data on that cellto fill in its piece.

Geom extracts from the database all data in its relations with the given cell name, reading one tuple at a time, creating it's internal representation. The internal representation is a Simula class-reference structure, one class for each relation, one class instance for each tuple. When it is finished, Geom returns to Desktop.

Essentially, Geom uses the relations as forms to hold the data from each Simula class. The full power of the relational database is not used, because Geom does no processing of the data with which the database might help. Other applications, such as design rule checkers and even the Database to CIF2.0 conversion program might use the relational database operations.

9.0 "Back Door" Data Entry

Geom runs in the Desktop system and allows other editors in that system to alter its internal data. This interface makes Geom easily extensible as new representations can be written which create cells and data for Geom.

SD, the Stick diagram editor in Desktop [2] creates Geom data. Therefore, users of the combined system can create data in the stick editor and have stick diagrams automatically fleshed out to mask geometry, or they can produce the mask geometry directly with Geom. Individual cells can be designed with either system and their mask geometries combined without concern as to the origin of the data. Cells can be originally created with SD, then altered or optimized by Geom.

The interface to Geom is straightforward. Other representations (editors) in the system can create new instances of the Simula classes that represent Geom's data elements: Box, Polygon, Wire, Instance, Cell and Connector. Each class has a number of procedures in it which alter its internal structure. So there is no worry about other representations creating bad data types, the legal types are the classes. Each of the data elements knows how to modify itself, so there is no problem with inconsistent internal state of the data. It is still possible to create strange elements, such as self-intersecting polygons and degenerate boxes, but it was felt that instead of restricting the elements themselves, Geom ought to restrict the user inputs, but allow the data to contain anything. Thus, Geom is compatible with all other systems and accept their unrestricted input, yet it is also an efficient and simple mask geometry editor.

Boxes and Polygons cannot be created or manipulated by Geom through the user interface. However, other representations, such as sticks and a layout language representation, need not labor under the constraints imposed on the user interface.

Therefore, these representations can create any feature on the mask. The features they create can still be manipulated by Geom, although manipulations of boxes and polygons are limited to Move, Copy, Delete, Rotate, Mirror and Scale.

Geom may eventually use back door entries onto other representations' data. Since the connectivity of the cell is apparent in the wires, a logical extension to Geom is to compute a netlist as the data is entered. A netlist is not really a proper data item to include in a mask Geometry editor. Some other editor should provide the data structure for that data, optimized for the operations it must perform. Geom should then have the ability to enter data in the "Wiring" representation.

10.0 Discussion of the Implementation

The Geometry Editor is written is Simula [16] and resides in three files: GREP.SIM, GEODAT.SIM and GEOM.SIM. GREP contains high-level general classes for the editor. GEODAT contains the classes which make up the data of the editor, Class Wire, Instance, and so forth. GEOM contains the command handling procedures for the editor and the editor initialization.

10.1 Data Structure

The data is represented internally as Simula classes. The classes have some data and some procedural attributes. The data attributes personalize each instance of the class. Classes can be "subclasses" of other classes. Subclasses inherit the attributes of the "superclass".

The main class in the database is the class geomcell. Geomcell is pointed toby the Desktop cell object, which points to cell pieces for other representations as well. Geomcell has data attributes to describe itself: its bounding box, connectors and a vector list of contents, the graphic elements that make it up.

The cell also contains variables to save some of the editor state that is particular to that cell. If the user leaves a window onto a cell, the Mark, selection and grid size are saved in the geomcell instance. They are particular to that cell while it is being edited, so all windows onto the same cell in the Geom representation have the same

Mark, selection and gridding characteristics. Different cells have different values for these variables.

Each graphic element, ginstance, wire, box and polygon, is represented by a class. They are all subclasses of elem, so they inherit elem's attributes: nbb, for minimum bounding box; show, to display themselves on the color display; erase to remove themselves from the display; contains, to see if a position is inside the element; deselect, to show the element unselected; and Xfrm, to apply a transformation to the element. These basic operations are done by the graphic elements themselves. For example, the Move command simply tells the element to erase, Xfrm by the distance of the move, then show. The move command does not even know what kind of graphic element it moved, it does not need to know.

10.2 Code Structure

Geom is built on the Desktop system, which is built on the Graphics packages [15], which is built on the low-level Things package (See figure 2).

Control always starts in Desktop. Activity is always initiated by a mouse event. Mouse events outside all windows are interpreted by Desktop. Typically, these are Desktop menu requests. Mouse events within a window cause the window to be brought to the top to allow editing if it is not already on top. If the window is on top, Desktop calls a procedure in the editor in the window. The editor then inspects the keystate, mouse state and cursor position and decides which command to execute. When the command completes, the editor must returns control to Desktop by leaving the procedure.

All transfers of control from Desktop to the editor are in the form of procedure calls. There are procedures for initializing the editor, refreshing the window, reading and writing database files and printing on the plotter. In short, there are procedures for all the system level tasks which are specific to the representation. Desktop does not care how the editor shows itself, for example, but it is agreed implicitly that the editor will show itself in some fashion when the procedure show is called.

Some of the Desktop communication procedures are handles by GREP. These include initializing the editor and first handling of the menu command. There is a convention

among the Desktop users that the header area contains an editor dependent menu. Selections in this menu are handled initially by a procedure in GDEP, then passed to GEOM if the command is not understood. Geom inspects the menu selection and branches to the appropriate menu procedure.

If the mouse event was in the data area of the window, control passes to the bug procedure in GEOM, which branches on the keyset and mouse state. The proper command handling procedure takes control.

10.3 Algorithms

Many of the algorithms used, especially those on wires, are not common lore, so, to give a better idea of what operations actually take place on the data, the following description of some of Geom' algorithms is provided.

10.3.1 Select

One piece of state that exists in the cell is the selection. Selected elements are operated on by most commands. The selection algorithm is designed to give the user freedom to easily select one element or several. When the select command is given, a rectangle is formed from the points of the mouse downstroke and upstroke. If that box is totally contained within an element or if the rectangle totally contains the minimum bounding box of an element, then that element is selected.

Thus we have one command to select a single element and to select all elements in an area. If the user points to an element, then the selecting rectangle will be very small. The element pointed to will be selected. The rectangle will be too small to contain any other elements, so only the specified element will be selected. If the user draws through an area, no element will be big enough to contain the rectangle, so only elements in elements in the area will be selected. The two uses of the command seldom overlap, and if the wrong items are selected as a result, the user can try again. Such is the beauty of an interactive system.

There is no select single item. If the cursor is inside two elements, they will both be selected. This is seldom a problem, as most elements have some area of non-overlap. If there is no non-overlap area, then one can select an area inside the overlapping

elements but around the desired one. Problems occur only with co-incident features, which are a problem in any case.

More complex selections can be built using the "add to selection" feature. Usually, the selected elements will be de-selected when the select command is given, but there is a Select More command which does not de-select elements. The test for inclusion in the selection is the same with the Select More command.

10.3.2 Select Points

The select points command accepts all points within the rectangle drawn by the user. Select points accepts points only on selected elements, so a subset of points in an area can be selected. Only points on wires can be selected, boxes and polygons are immutable. The selection de-selects selected instances, boxes, polygons and wires that had no points in the point selection. The wires with points selected are still selected normally and can still be manipulated in toto as selected items.

10.3.3 Undelete

When elements are deleted, they are moved into a buffer in Geom. These elements can be un-deleted with another command. The elements are brought back to their same positions relative to the Mark. Elements can be deleted in one cell and undeleted in another. Elements can be un-deleted only once.

10.3.4 Chop Cell Boundary

When defining a cell's parameters, the user may specify a bounding box for the cell. Nearly all data outside the box is chopped off and left in the cell, selected. They can be deleted immediately with the delete command. General clipping would require that wires be converted into polygons (figure 6). Instead, the clipping algorithm retains the type of each element: wires clip to wires, and so forth.

Instances, boxes and polygons are not clipped. They fall outside if their bounding box extends outside the new bounding box of the cell at all.

Wires are clipped as shown in figure 7. The centerpaths of the wires are clipped so

that the midpoint of the true end of the wire touches the border. Wires which pass through the edge at right angles are clipped so they do not extend outside the bounding box. Wires that pass through the edge at an arbitrary angle are not clipped so cleanly. Part of the wire extends outside the box and part does not extend all the way to the border, but a corresponding angled wire in a butting cell would fill the gap properly. It is assumed in this situation that the user is doing something clever andhe can watch out for himself.

Another special case occurs when a wire segment is parallel to an edge of the bounding box within half the width of the box. Since the centerpaths of the wires are clipped, if the center path is inside, then the whole segment is kept, and part of the area of the wire extends outside the box; if the centerpath is outside, the entire segment is clipped off. Clipping is not true clipping, so that wires may be retained as wires.

When a wire segment is split by the clipping algorithm, the two wires thus created have a common endpoint. This causes the display to look strange when the chaff is deleted, but a refresh will clear up the screen. More importantly, though, if the clip was not satisfactory, the wires are set up to be welded together again without moving any pieces.

10.3.5 Snap Points

Points on wires are snapped on input so that wire segments are horizontal vertical or diagonal. This is not a severe restriction of design freedom, and provides the designer with the ability to point to a position near where he wants the wire to go and have the position corrected. As seen in figure 8, there are a number of legal points which satisfy the constraint just mentioned. The Geom snapping algorithm moves the input point either horizontally or vertically (not diagonally) to a horizontal, vertical or diagonal line from the last point on the wire. The legal point which is nearest the input point is used.

10.3.6 Move Points

The move points command can be used to re-position many points on a wire. However, the user might specify a new location for the points that violate the forty-five degree

rule. When positioning those points, the move points command applies the wire snapping procedure described above. Each point on each wire is snapped independently, starting with an endpoint of the wire which was not moved. So, when moving a block of elements, they might not move as a unit, each will move with its own constraints. Also, wires will change shape, sometimes drastically, to preserve the angles. The worst part of this is that points may move which were not in the point selection. This would certainly surprise the user, probably not in a favorable fashion. This problem is discussed in the section "Limitations".

10.3.7 Stretch Width

The wire width change command corresponds to a transverse stretch of the wire. Geom assumes that the user is trying to stretch a wire similar to the way one would stretch a box. Therefore, the near edge of the wire moves to intersect the cursor position and the far edge of the wire does not move. The centerpath has to move to accomplish this, as a simple width change is not sufficient. The algorithm proceeds as follows:

The perpendicular to each segment to the cursor location must fall on at least one center path segment. The smallest distance from the cursor to any acceptable segment is computed. This plus half the old width is the new width, as can be seen in figure 9. Half the difference between these two numbers is the distance the path must be moved.

The centerpath is relocated by moving the first segment on the wire, then for each successive segment, finding the parallel to the segment that is displaced properly and intersects the previous new segment. The endpoints of the wire are not moved, so the true ends of the wire change position. This is a cause for concern as it could lead to inadvertant design rule violations. However, it is hard to correct, because, if the segment is very short, it could be widened sufficiently that the endpoints would have to be moved backward past the previous point on the wire. Thus the shape of the wire would change. This problem is discussed in the section "Limitations".

11.0 Limitations

One feature that has been found to be useful is the ability to expand an instance in place, copying the data from the instance to the containing cell. This feature has not been provided in Geom, and this might force some designers to re-enter complete cells because they cannot expand an old one. This problem is alleviated somewhat by the ability to delete in one cell and un-delete in another.

Geom forces bottom up design because there is no way to make an instance of a cell which does not already exist in the Desktop system, and while instantiating, make an estimate of the size of that cell. Gary Tarolli's architecture editor [14] has this feature. Such a feature should be in Geom as well. Perhaps it should be in Desktop. In the meantime, the user can enter all his cell names in the Desktop browser in advance, then proceed top-down, opening a new window onto a cell when necessary to define a bounding box, then returning to his higher level cell.

As mentioned in the discussion of the implementation, when performing a move points command, the points are all moved independently. Instead, they should be moved as a block, retaining their relative positions. This probably will not be a problem, as most moves are very easy cases, but when this problem does occur, it can damage the design severely and without warning. Moving points as a block has its own problems. Each point would add its own set of constraints to the move. Finding a legal position might be time consuming. There might not even be a legal place to put the block if the features involved were created by some other representation and did not obey the constraints of the user interface. Something should be done about this, though, because non-selected points must not move or the connectivity could be inadvertantly destroyed.

Another addition to the move points command is to have whole boxes and polygons be movable. This would make the move block feature especially useful, as the designer need never concern himself with the types of elements in the block he is moving. But this is needless complexity. The only way boxes and polygons can enter the database is through another representation anyway. The other representation should position them properly.

When widening a wire, the ends can move, as mentioned in the discussion of the algorithm involved. This should not be so. A method to avoid the problem of changing

the shape of the wire is still under consideration. One idea is to not move a point past the neighboring point in the wire. This will cause the stretch to work properly in most cases, but wires will change strangely in other cases. Another idea is to allow the wire to change shape, expecting this to cause problems less often than the moving ends. Unfortunately, both these operations are not reversible, and this irreversibility occurs in precisely those situations when the effect was unexpected.

The user cannot see the grid on which his points fall unless the tick marks are turned on, but the tick marks tend to clutter the screen. It would be nice if there was code for the LSI-11 to show the cursor only at grid points. This would let designers know clearly which grid point the cursor was on, eliminating confusion that exists now, causing the move command to be given several times to do one move operation.

Geom provides no numeric feedback for the designer. There are no facilities except the grid for measuring distances between features on the screen. Some feedback, such as the position of the Mark and the distance between the Mark and its previous location are, if not necessary, then at least very handy.

The most serious limitation on the Geom is presented by the DEC-20 itself. The processor is powerful enough to run Geom interactively, but interaction suffers severely when several users attempt to use the machine simultaneously. When the quick feedback disappears, Geom becomes very hard to use. The command structure of Geom was designed to allow the user to quickly try ideas in the design and not incur a penalty for experimentation. When the performance degrades, there is a very serious time penalty to be paid for each command, and the editing becomes very frustrating. This problem could be resolved with more memory on the DEC-20.

12.0 For the Future

Geom handles only six layers at present. Requests have already been made for eight layers. There are problems with this, though, because the color display has only four bits per pixel. It is difficult to show the six layers already in Geom without ambiguity. Eight layers could be done, but, in some cases, the designer would have to "turn off" the display of some layers to keep them from interfering. It may be that we have reached the limit of the four bit color display.

Another suggestion has been to do design rule checking on the fly as data is added. This would ease the design rule task, because searches can be done on smaller pieces of the design. Also, the designer could get instant feedback on the correctness of his design, drastically shortening the design turnaround time.

Since wires preserve the connectivity of the cell, one would like to interpret the structure of the cell on the fly. This is currently not done in Geom for a number of reasons. At present, though, there is no data structure for the wiring data. This data structure should be provided by some other editor in Desktop system.

As mentioned at the beginning of this thesis, Bristle Blocks users could very definitely benefit from an mask geometry editor. Some I/O features and Bristle Blocks specific commands must be added to make the geometry editor capable of producing blocks for Bristle Blocks quickly and easily. This would drastically cut cell design times and make Bristle Blocks a much more attractive tool.

An immediate goal for Geom is the addition of language features as described in [9]. Such a system would combine the best of both worlds: fast cell creation with the interactive graphic facility, and powerful parameterization with the language facility. The resultant system would be a quite powerful design tool.

13.0 Conclusions

The mask geometry editor is a valuable tool for the creation of integrated circuits. It was designed to be easy to use and powerful in applications calling for quick design of prototype LSI circuits. It was also built to be part of a larger whole, including not just the Desktop system in which it runs, but in the Caltech design community.

The use of wires as the single graphic primitive was a good decision. Wires provide a good model for the kinds of features on integrated circuit masks. The commands that deal with wires are powerful and not overspecialized. The ability to guarantee that the connectivity will be preserved under a number of operations seems to be a powerful selling point.

The major limitation in Geom appears to be the limited interaction from an overloaded

timesharing system. This limitation may be removed in the near future.

14.0 References

[1] John D. Williams, "Sticks -- A New Approach to LSI Design", Master's Thesis, Massachusetts Institute of Technology, June, 1977

[2] Jim Rowson, "SD: A Stick Diagramming System", Caltech SSP File #2802, May, 1979

[3] Dave Johannsen, "BRISTLE BLOCKS: A Silicon Compiler", Caltech SSP File #2587, January, 1979

[4] Charles Minter, "Third Version of a Color Display", Caltech Display File #2101, October, 1978

[5] Larry White, "Color Terminal Graphics Protocol, Version 3.1", Caltech SSP File #2584, February, 1979

[6] Jim Rowson, "DeskTop -- An Editor Building System", Caltech SSP File #2198, November 1978

[7] Douglas Fairbairn and James Rowson, "ICARUS: An Interactive Integrated Circuit Layout Program", Proceedings of the 15th Annual Design Automation Conference, June, 1978

[8] R.F. Sproull and R. Lyon, "The Caltech Intermediate Form for LSI Layout Description -- Revised --", SSP File #2686, May, 1979

[9] Stephen Trimberger, "A CAD system Combining Interactive Graphics and a Layout Language", SSP File #2499, February, 1979

[10] Stephen Trimberger, "GEOM - Geometry Editor Functional Specification", SSP File #2804, May, 1979

[11] Jim Rowson, "SSP Data Base: Not Just Data, Reality", SSP File #2500, February,

1979

[12] Mike Ullner, "RDP User's Guide", SSP File #2803, May, 1979

[13] Larry White and Stephen Trimberger, "Maximum Bounding Boxes", Display File #2684, April 1979

[14] Gary Tarolli, "APE Functional Specification", SSP File #2801, May, 1979

[15] John Wipfli, "A SIMULA Graphic Package", SSP File #1929, October, 1978

[16] G. Birtwistle, O.J. Dahl, et al., "*Simula Begin*", *Petrocelli/Charter*, 1973

[17] Hewlett-Packard, "Operating and Service Manual 7221A Graphics Plotter", HP 07221-90000, Aug, 1977

[18] Gary Tarolli, Master's Thesis, SSP File #2819, May, 1979

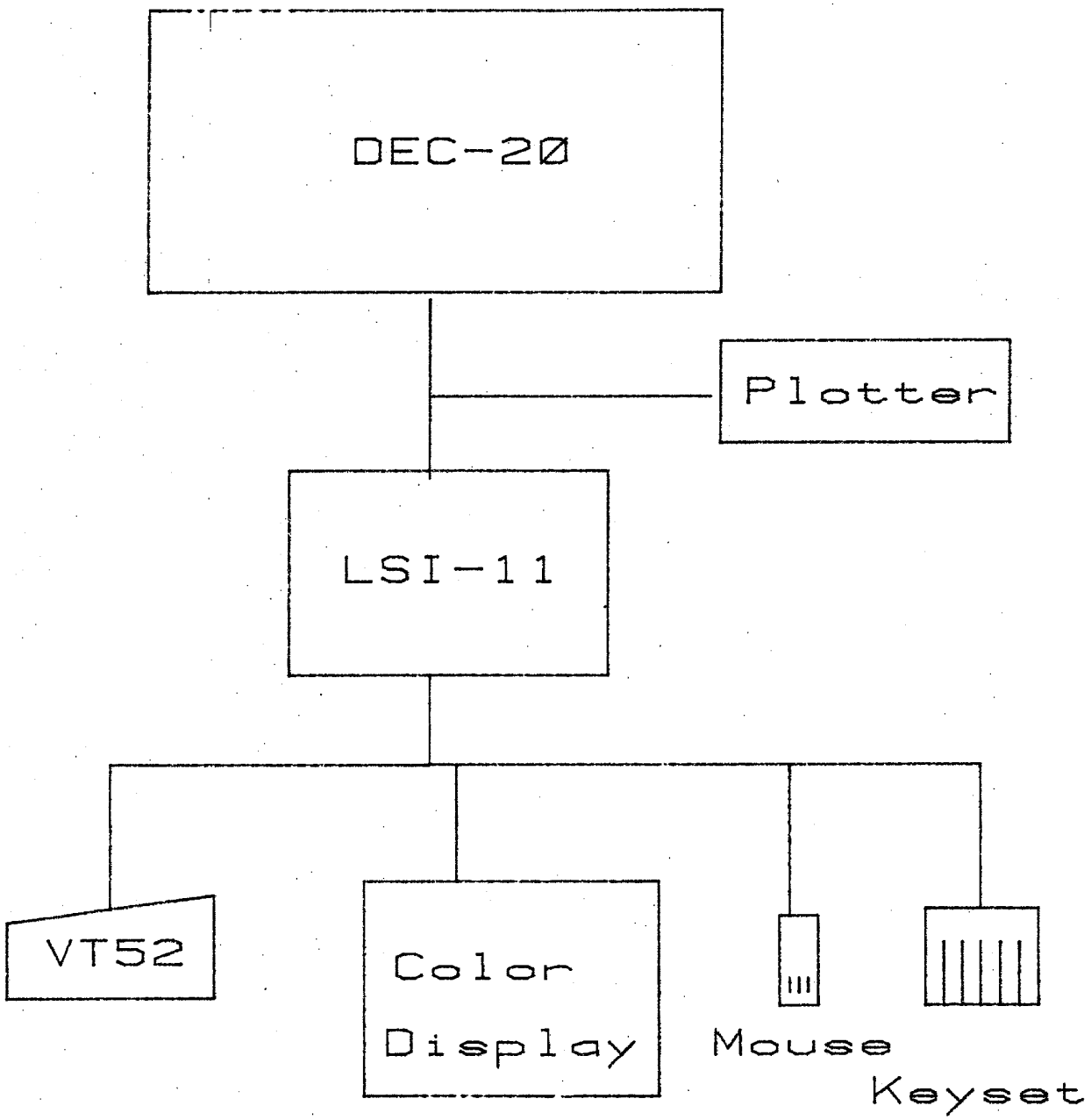

Figure 1. Hardware Organization

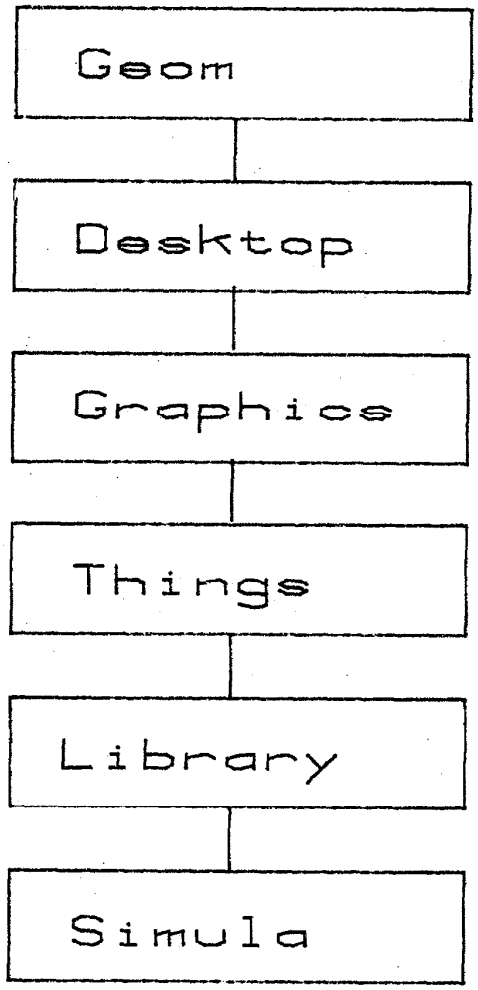

Figure 2. Software Organization

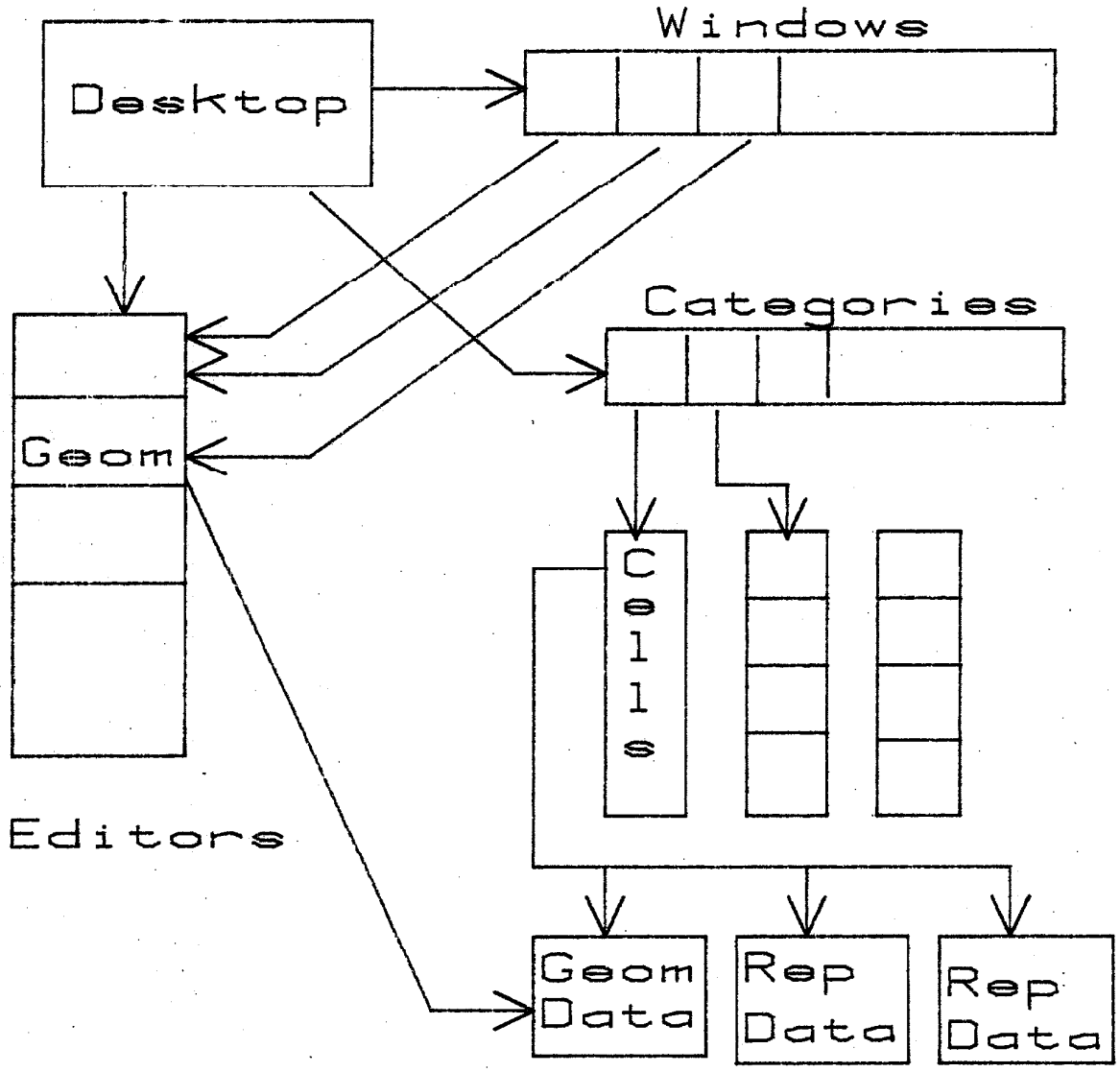

Figure 3. Desktop Class Structure

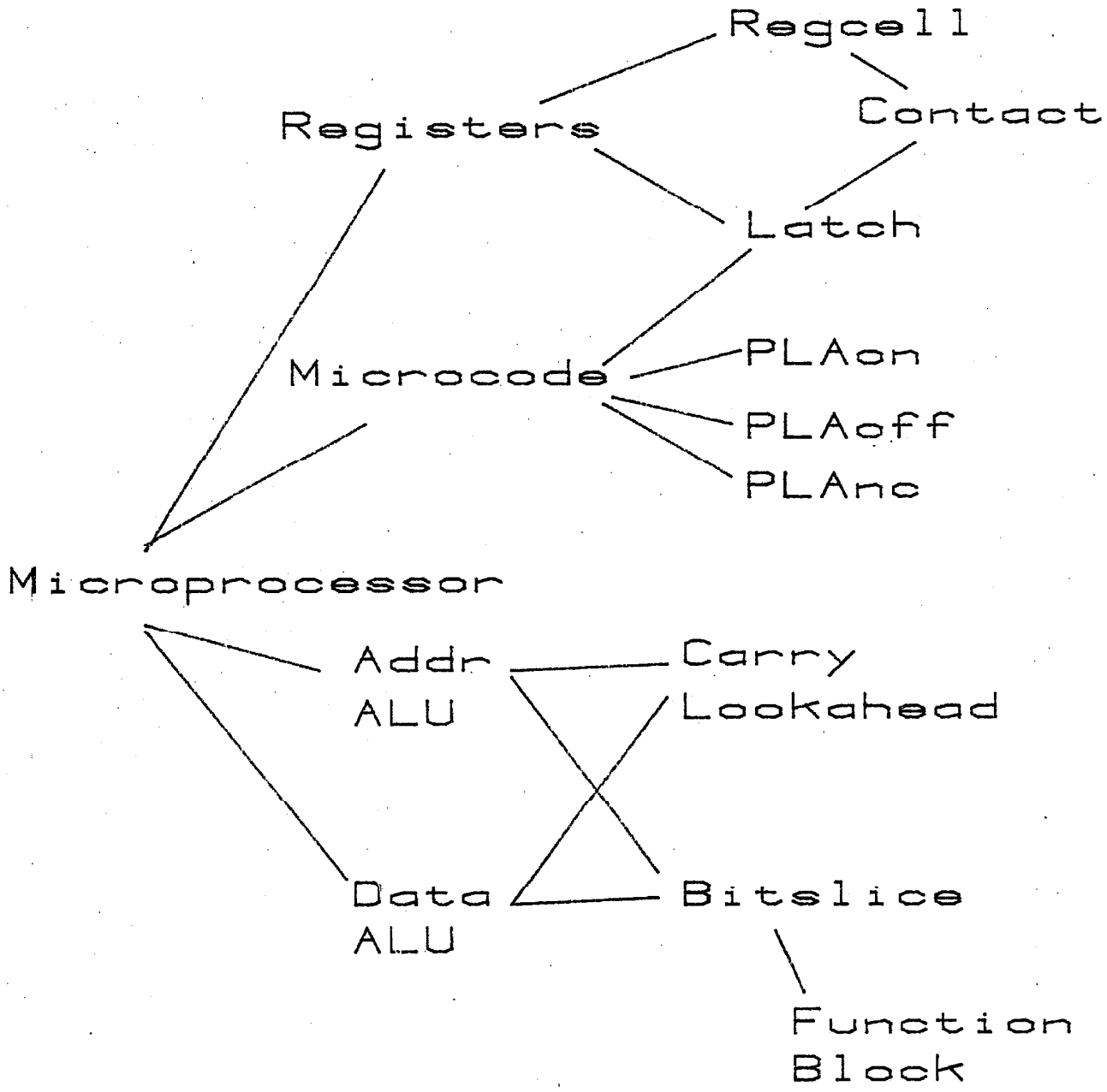

Figure 4. The Design Hierarchy

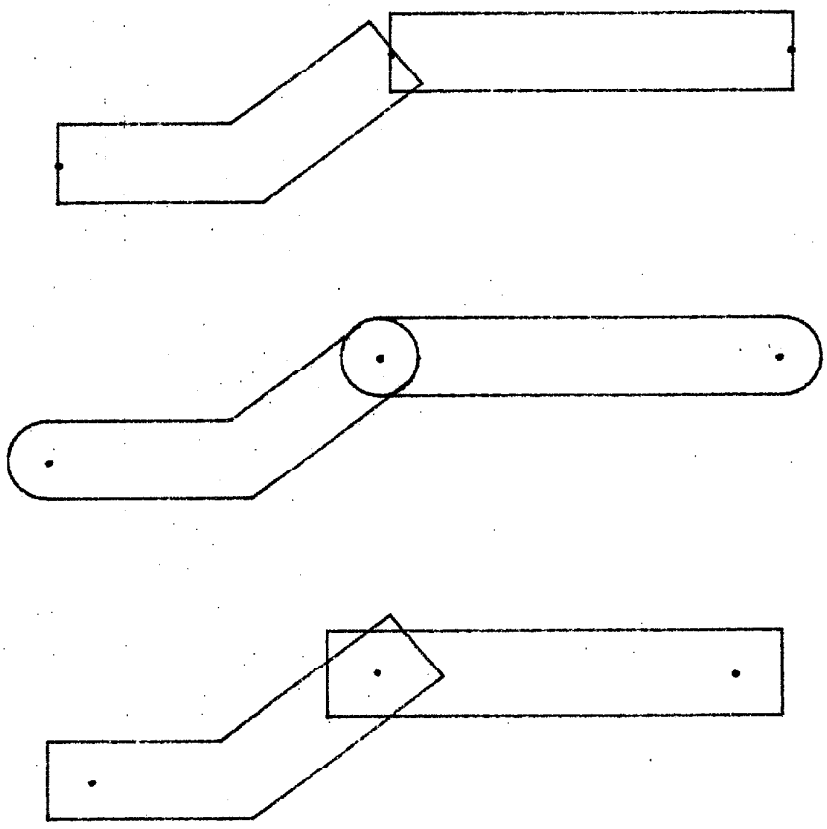

Figure 5. T

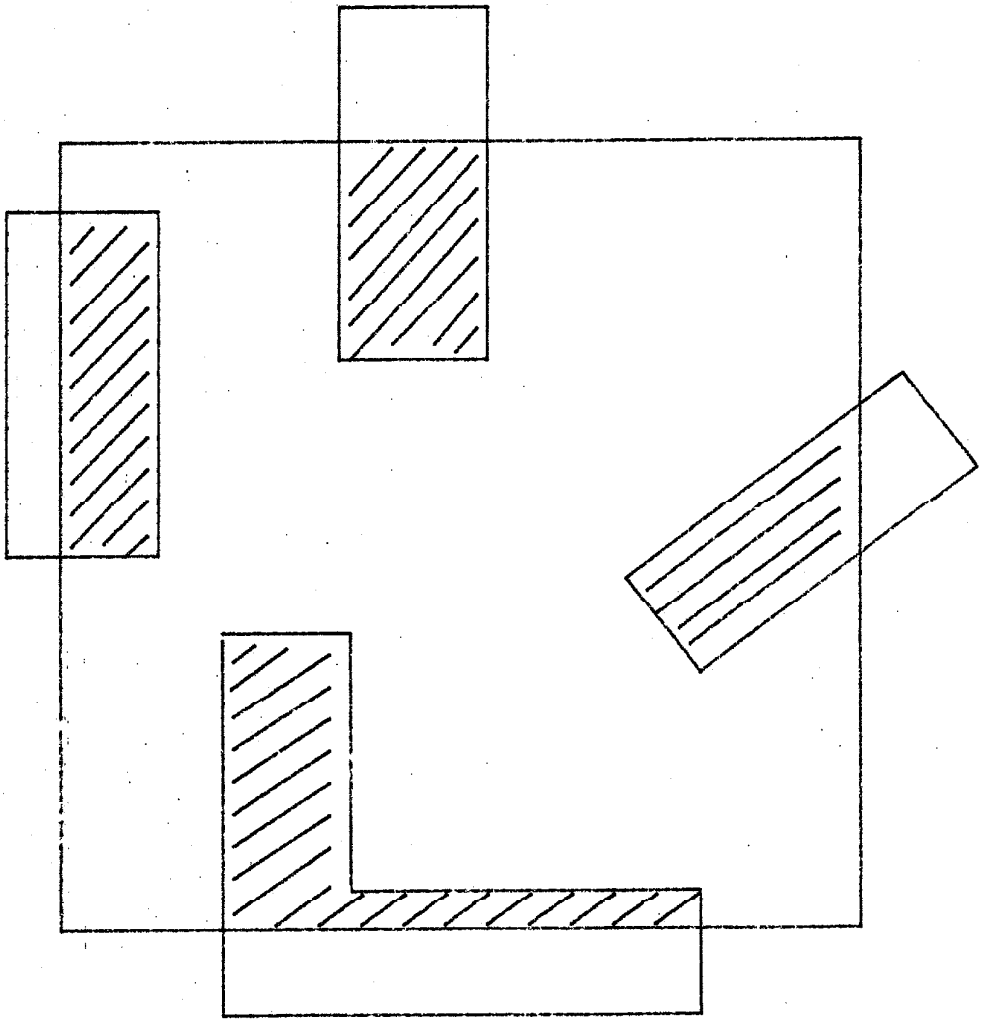

Figure 6. General Clipping Yields Polygons

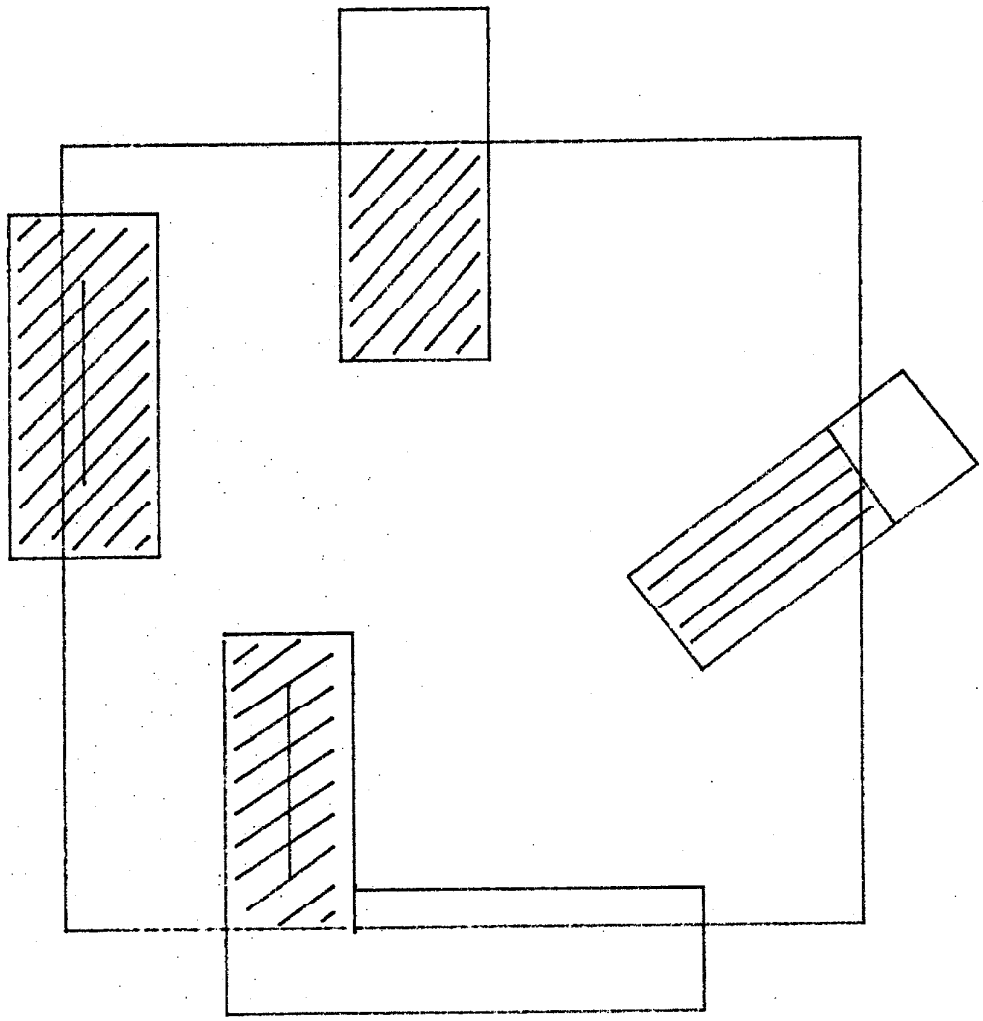

Figure 7. Clipping Center Paths Yields Wires

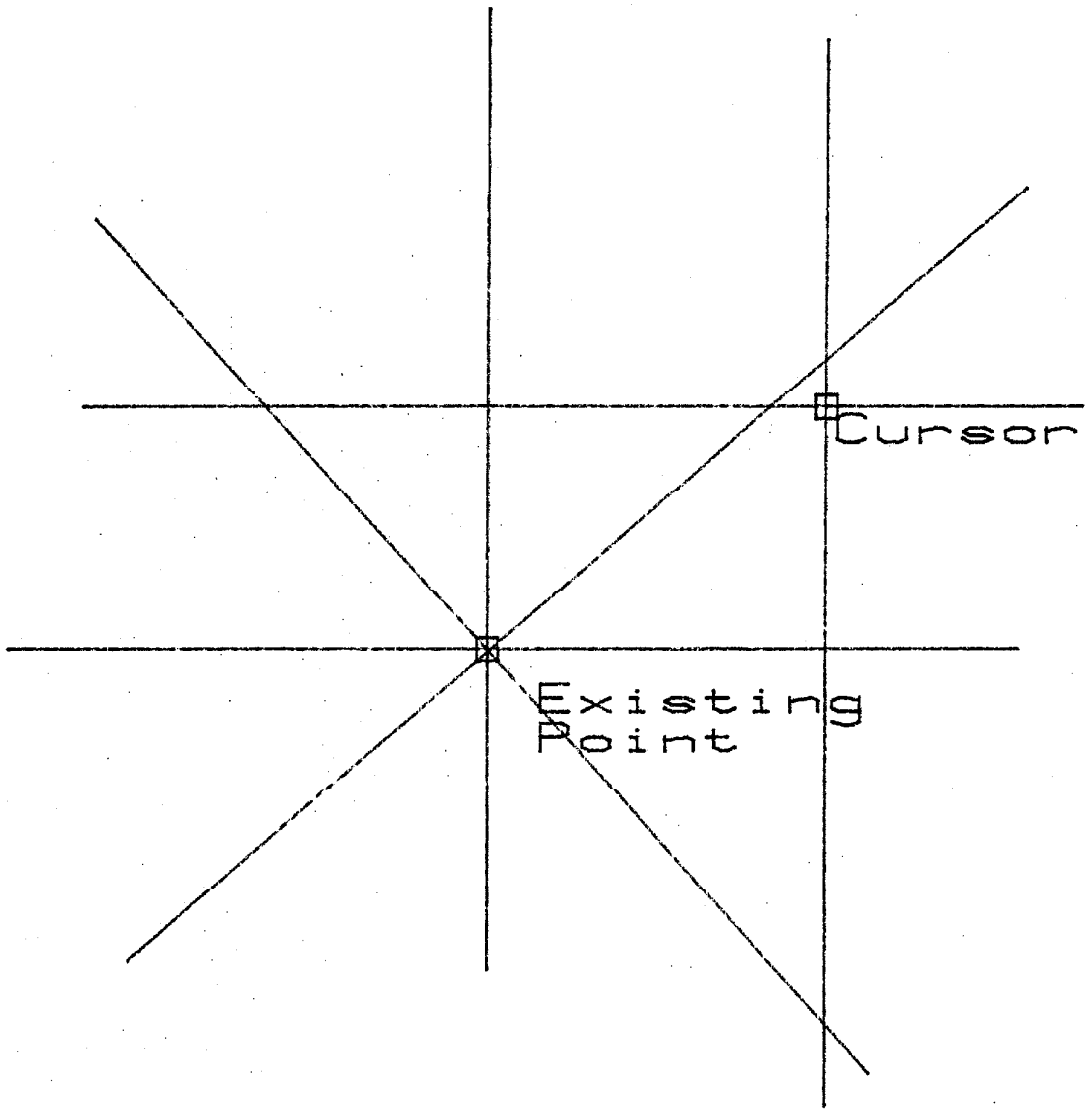

Figure 8. Points Which Meet Horizontal, Vertical, Diagonal Constraint

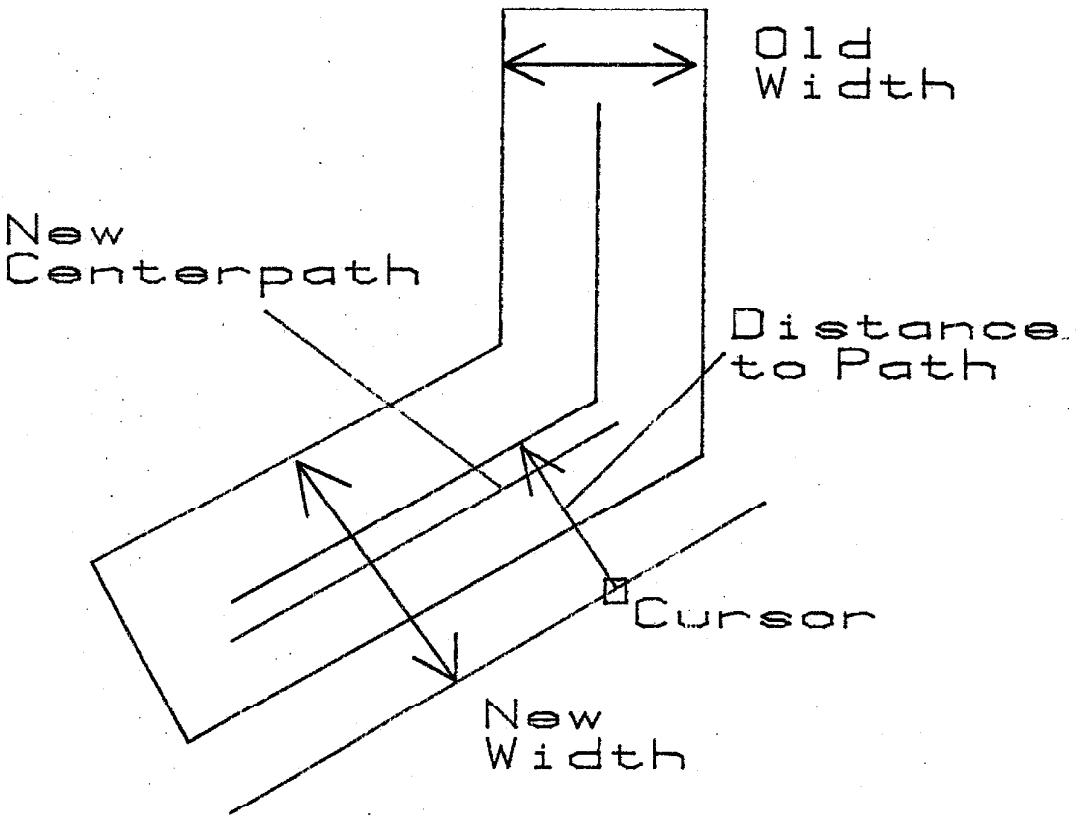

Figure 9. Transverse Stretch of Wire Width

Geom User's Manual

Geom is an integrated circuit mask geometry editor. It was designed to facilitate the task of creating prototype and experimental LSI circuits. Its users are expected to be system designers with a working knowledge of integrated circuit design.

Geom is an editor in the Desktop system. It can be used either as a target representation for other representations in the system or as a stand alone IC graphics editor. The latter application will fill the bulk of this document, the former will be discussed in a section at the end.

Hardware Environment -- The Color Terminal

Geom runs on the Charles color terminal. The terminal is connected to a VT52 terminal and output can come to either display. The Charles terminal is driven by a LSI-11. The display is approximately 480 by 512 by four bits per pixel. Four bitsis enough to do four layers and their overlaps. Geom handles six layers with a minimum of confusion.

Graphic input devices consisting of a Xerox mouse and keyset are handled by the LSI-11 also. The mouse has three buttons which shall be referred to in this document as Mark, Create, and Select, from left to right or top to bottom. The keyset has five keys; from left to right: Delete, Move, More, Edit and Scroll. The uses of the mouse buttons and keyset keys will be discussed with the individual commands.

Software Environment -- Desktop

Desktop provides windows in which the editors reside with a cell and handles the creation, destruction and manipulation of those windows. There can be as many windows as the user wishes, looking at the same cell or different cells, in the same representation or different representations.

Only one window is active at a time. To activate a new window, the user must click a mouse button in that window's visible area. That window is then brought to the

top and refreshed.

Desktop has one permanent window, the browser, which provides facilities for the creation and organization of cells. The browser window has three panes. The first pane contains categories of cells. New will allow a name to be typed on the VT52, and will create a new category with that name. The middle pane contains cells in the currently selected category. New in this window allows the user to name a new cell in that category. The cell is empty and can be edited with the editors in the Desktop system. The third pane contains editor names, also called representation names. The user cannot create new representations, he must use what has been provided. When a cell and a representation have been selected, the New at the top of the browser can be selected. The viewing window is then specified by drawing through a diagonal of the window with any mouse button held down. If the representation selected was Geom, then the rest of this document is relevant.

Desktop has two menus of interest, the window menu and the system menu. The window menu has commands for the current window: Print, Close, Frame, Push, Pop. The system menu has commands of interest to the Desktop system: Quit, Debug, Save and Get.

For a complete discussion of Desktop, see SSP File #2198, "Desktop, an Editor Building System".

The Editor State

Geom has been built as much as possible to be a modeless editor. All parameters for a command are already present when the command is given. Although this is not always the case, particularly with those commands requiring typed input, it is usually the case. For this reason, the editor has many pieces of state which are used as arguements to commands. There are commands for manipulating the editor state independent of the commands which manipulate the data.

The Mark shows on the screen as a small white cross. It simply marks a point which may be used in a command. For example, when the user creates a wire segment, he needs only to specify one point of the segment, the other is implied to be the mark.

The mark is moved to the cursor position by pressing the first button on the mouse. Many commands move the mark automatically in order to position it at a good next position. Thus the mark may be automatically positioned correctly for the user, saving time and effort in the editing session.

The Selection consists of a set of graphic elements: instances, wires, boxes and polygons. The selected elements are shown on the screen with a white centerline, for wires, or a bold white outline, for the others. Commands such as Move, Copy and Delete operate on the selected elements. There is no implied commitment to use the selection, it merely exists. As with the Mark, some commands leave elements selected which may be helpful.

A second part of the selection is the point selection. It is possible to select a set of points from wires and move the points or break wires at those points. Points selected in this manner are shown with small white dots. Points on boxes or polygons and points in instances cannot be selected.

The editor remembers the last deleted elements. These may be un-deleted if desired. If not, they are merely discarded when something else is deleted.

When creating a wire, the editor remembers the current wire, the wire that will be extended by the next "Create Wire Segment" command. Nearly all commands unset the current wire. Leaving the window also ends the current wire.

The editor remembers the current default layer, the layer on which a wire will be created when the user gives the wire create command. Each layer has a default wire width.

There is also some state associated with the window: the magnification, the X-Y position of the center of the window, the size of the grid on which points are taken and the presence or absence of tickmarks.

Some pieces of state are carried with the cell, and apply to that cell no matter which window it is in. These are the mark, the selection, the grid size and the tickmark boolean. Each cell has its own copy of these parameters. Each window has its own magnification. The Geom editor as a whole has the current menu and the delted

elements.

Wires

The only graphic primitive available in the user interface is the wire. A wire consists of a path with a width and a layer. Wires extend half the width past their endpoints. Wire segments must be vertical, horizontal or diagonal. As points are added to a wire, they are snapped to ensure legal angles.

There is no limit on the number of points on a wire, or on the path of a wire. Self-intersecting wires are allowed, but they will look strange because of the plotting algorithm. Since a wire extends half the width past the end, a square flash can be made by putting both wire points at the same spot. There is no such thing as a one-point wire, because the "Create Wire Segment" command uses the Mark and the current cursor position as the first two points when a new wire is started. It is not possible to make a wire with any dimension smaller than the width.

Although the user interface provides no facilities for creating boxes and polygons, other representations in the Desktop system can create them directly in the Geom data segment. Commands that operate on whole elements operate on boxes and polygons as well as wires and instances. These are the general editing commands: Delete, Move, Copy, Mirror and Rotate.

Command Input in the User Interface

The main goal in the user interface is the obvious goal for an interactive system: to make a command structure that is easy to learn and fast to use.

Commands are separated into two categories, "fast" commands and menu commands. Fast commands use the mouse and keyset combination to allow highly-interactive commands to be given with a minimum amount of user overhead to allow faster editing. All commands are initiated by a mouse event. When the cursor is in the window header, the mouse event is used to pick a menu item, when the cursor is in the data area of the window, the mouse event in conjunction with the keyset state is

used to determine which fast command is to be carried out.

Fast Command Input

The keys on the keyset are used as modifiers to the three mouse buttons. A command is given by holding down the corresponding key and pressing the proper mouse button. For the sake of simplicity, there are no multiple key chords. Thus, there can be up to eighteen fast commands, including those with no keyset input. Not all are used.

The three mouse buttons have different emphases. The Mark button is associated with small changes of the editing state. The Create button usually results in data being added to the database, and the Select button changes the selected element or elements.

Each keyset key has a different use. The Delete key is used to remove elements, the Move for moving and copying elements. The More key is used to add elements, the Edit key for wire editing commands, and the Scroll key for moving the viewing location. The precise functions of combinations of mouse buttons and keyset keys will be described with the individual commands.

The fast commands are Mark, Draw Segment, Delete, Undelete, Move, Copy, Select, Select Additional, Select Points, Move Points, Widen Wire and Scroll. Because of their frequency, Mark, Draw Segment and Select require no key, only a mouse click.

Menu Command Input

A menu command is given by MARKing the entry in the menu corresponding to the command. Some menu commands are carried out immediately, some require more information. Commands which require more information prompt the user with a message sent to the VT52. The additional information may be a number or name to be typed on the VT52 or it may be a mouse event. When the menu entry is marked, it will show "selected". Selected layer entries show outlined, selected text entries turn blue. When the command is finished, the "default" menu entry will be

automatically selected. For the main menu, the default entry is the current editing layer; for the other menus, the default entry is the return to the main menu. Automatic resetting of the menus carries out no action, it is merely to let the user know when the command has finished.

The menu commands are split into three menus. The main menu contains the layer changes, the current layer default wire width change, layer on/off, and select keys for the other two menus, Mod and Xform. The Mod menu contains the following commands: ToCIF, Tickmarks, Create Instance, Instance Visible/Invisible, Cell Parameters, Change Grid Size, Weld, Extend, Break and return to Main menu. The Xform menu contains Scale, MirrorY, MirrorX, Rotate and return to Main menu.

Changing the Editor State

In the discussion of the commands, the command is indicated in parentheses. It is either a menu name {Main, Mod, Xform} followed by the menu entry (i.e. Mod: Weld) or a Mouse and Keyset combination, (i.e. CREATE & Delete). Additional data is specified by name and type, (i.e. width: Real).

(MARK) Mark. The Mark is changed by pressing the MARK button while the cursor is in the data area of the window. The Mark is moved to the nearest grid point to the cursor position.

(SELECT) Select Elements. Select Area. The select button by itself will select all elements totally contained within the rectangle swept out by the cursor while the select button was depressed. In addition, all elements which totally contain the area swept out by the cursor will be selected. Thus, one command can be used to select an individual element, by positioning the cursor inside it and pressing Select momentarily; and for selecting an area, by moving to one corner of a rectangle surrounding the area, pressing Select, moving to the diagonal corner and releasing Select. The previous selection is de-selected. Notice that this command selects all elements that meet the criteria given above. There is no guarantee that there will be only one selected element.

(SELECT & More) Select Additional Elements. Select Additional Area. A selection

normally de-selects previously-selected elements before selecting the new elements. If the More key is held down when the Select button is pressed, then the new selection will be added to the already selected elements. This allows the user to build up a complex selection.

(SELECT & Edit) Select Points. All points are selected which are on selected wires and are within the rectangle swept out between the downstroke and the upstroke of the Select button. Only points on wires that are already selected can be selected. The points thus selected can be moved with the Move Points command or used as positions in the Break Wire command.

(Main). Change Layer. The current layer is changed by MARKing the menu entry corresponding to the desired layer. The current layer is shown outlined. The main menu will change to show the new selected layer outlined and the default width for that layer in the width entry.

(Main: W) New width: Real. Change Width. The number following the W in the main menu is the default width for the current layer. When the menu entry is marked, a prompt will appear on the VT52 for the new width. The user must then type the new wire width for the current layer. The widths remain in effect until changed by the user.

(Main: See) Layer On/Off. This command toggles the visible boolean for the current layer. If the layer is off, elements on the layer cannot be seen, but they can still be selected. Making a layer invisible diminishes the clutter due to many overlapping or partially overlapping features on many layers. Part of the problem is due to the four bit per pixel limitation on the Charl[illegible] is the inability of the human eye to distinguish slightly varying color.

(Mod: #) New grid size: Real. Change Grid Spacing. The number following the pound sign (#) in the main menu shows the current grid size. Positions entered by the user are snapped to the grid points. When the pound sign entry in the Mod menu is marked, a prompt will appear on the VT52 for the new spacing. The user must then type the new grid spacing.

(Main: ::) Tickmarks. The tickmarks are turned on and off by MARKing the grid

mark, four white dots in the main menu. The tickmarks are positioned at the grid points.

Magnification. The white bar in the header of the window is the magnification bar, higher magnification to the left. To change the magnification, point to the desired position of the bar and push the Select button.

Commands Which Change the Database

(CREATE) Create Wire Segment. A wire segment is created by pressing the Create button on the mouse. If there is no "current" wire, a two-point wire is created from the Mark to the cursor position. If there is already a current wire, then the new point will be added to the end of the current wire. The new point is snapped to the nearest grid point that conforms with the convention that wire segments must lie horizontally, vertically or diagonally. The mark is moved to the new endpoint of the wire. Subsequent presses of the Create button will add more points to the wire. A wire is terminated by executing a fast command that is not a Create Wire Segment and not a Scroll. The current wire is left selected.

(CREATE & Delete) Delete. This command deletes the selected elements from the current cell. Once deleted, elements can be un-deleted either into the same cell or another cell.

(CREATE & More) Undelete. The last elements to be deleted are put back into the currently open cell. The elements are positioned relative to the mark. If the Mark did not move between the delete and the undelete, the elements will be put back where they were. This facility allows data to be copied between cells. A set of elements may be deleted in one cell, then a window may be opened to another cell, and the elements undeleted there, relative to the mark in the new cell. Elements can be undeleted only once. This command leaves the newly un-deleted elements selected.

(MARK & Move) Move. This command moves the selected elements. The selected elements are displaced by the difference between the cursor position and the Mark. The Mark is moved to the cursor position. The moved elements remain selected, so

the move command can be given again if the first move was not correct.

(CREATE & Move) Copy. This command copies the selected elements. The new copies are displaced by the difference between the cursor position and the Mark. The Mark is moved to the cursor position. The newly created elements are left selected, so that they can be moved, if necessary, to exactly the correct position.

Transformation Commands

(Xform: Rot) This command rotates the selected elements ninety degrees counterclockwise around the Mark. The rotated elements are left selected.

(Xform: MX) This command mirrors the X co-ordinates of the selected elements using the mark as the origin. The mirrored elements are left selected.

(Xform: MY) This command mirrors the Y co-ordinates of the selected elements using the mark as the origin. The mirrored elements are left selected.

(Xform: Scale) Scale factor: Real This command scales all elements in the cell (not just the selected elements). The scale factor must be typed at the VT52.

Commands That Modify Wires

(CREATE & Edit) Stretch Width. This command moves the nearest edge of the selected wire so that it passes through the cursor position. There must be only one element selected and it must be a wire. The point of intersection of the perpendicular to the segment through the cursor position must intersect some segment between the endpoints. The minimum distance thus acquired is used to make the move. Notice that on a square flash, the endpoints are co-incident, so it is very difficult to stretch the width. When the width is stretched, the near edge and the centerpath of the wire move so that the far edge does not move. Warning: when stretching a wire the width changes, so the distance the wire extends past the endpoints changes. This can inadvertantly cause design rule errors or broken connections. The Mark is moved to the cursor position.

(MARK & Edit) Move Points. The selected points are moved. The points are snapped one at a time to ensure that wire segments are vertical, horizontal or diagonal. If all the points on a wire are selected, this is equivalent to a Move for that wire, if only a few points are selected, the selected points can be pulled away from the non-selected ones, stretching wire segments. The Move Points command can be used to do a "block move", to move a piece of the design. The wires will move where appropriate and stretch where appropriate. All points are re-positioned independently, though, so the block does not behave exactly as a block, but errors due to this behavior can be corrected by simply giving the Move Points command again. Warning: if interior points on a wire are moved, unselected points may be re-positioned. This will only happen in rather baroque edits.

(Mod: Brk) Break Wire. This command breaks the selected wire at the selected point. There must be only one element selected and it must be a wire. There must be only one point on that wire selected. The wire will be broken at the selected point into two wires. The two wires may then be manipulated independently. The two wires share a common point, so they may be easily Welded back together. Both wires are left selected.

(Mod: Weld) Weld Wires. This command welds two wires. There must be only two elements selected. They must both be wires on the same layer, have the same width and must have a common endpoint. The two wires will be merged into one, which can then be manipulated as a unit. The welded wire is left selected.

(Mod: Ext) Extend Wire. This command is used to extend the selected wire. There must be only one element selected, it must be a wire. The end closer to the Mark becomes the last point on the wire, so subsequent Create Wire Segment commands add points to that end. The mark is moved to the endpoint of the now-current wire.

All fast commands except Create Wire Segment and Scroll terminate the current wire. Create Instance, ToCIF terminate the current wire and exiting the window terminates the current wire.

Cells and Instances

(Mod: [ ]) Create Instance. The square in the Mod menu instantiates a cell with the origin of the instance at the Mark. The cell to be instantiated is the cell in the window immediately under the current window. If the browser is immediately under the current window, the selected cell in the browser is used. The origin of the cell is positioned at the Mark. The instance thus created must be manipulated as a single object. In order to manipulate the internals of the instance's defining cell, a window must be opened onto that cell. It is possible to make a recursive instance, but this should be avoided, as there is no meaningful way to interpret recursive cell references. There will be no problem with display unless all the instances in the recursive chain are made visible. Then an infinite loop will occur refreshing the display. There is no easy way to recover from this loop.

(Mod: Vis) Instance Visible. An instance can be shown on the screen either visible or invisible. This command toggles the visibility flag of all instances in the selection. When "invisible", the cell is shown by drawing the bounding box and the connectors. When "visible" the cell is shown by drawing the bounding box and all the data in the cell.

(Mod: Parm) Bounding Box Corners: Two Points; Origin: Point; Connectors: Rectangles. Set Cell Parameters. This command changes the cell parameters. The cell has three parameters: a bounding box, an origin, and connectors. All changes to the cell parameters are made with the Create button. The Select button is used to terminate the command states. All fast commands except Create Wire Segment and Select still work. Any menu command will abort the parameter change. Any changes made before an abort are there to stay.

During the bounding box setting, each time the Create button is clicked, the nearest corner of the box is moved to the cursor position. During the Origin setting, the Create button sets the origin to the cursor position. The Mark also is moved to the cursor position. The rectangles for the connectors are made by drawing through the rectangle of the connector between the downstroke and the upstroke of the Create button. The layer of the connector is the layer of the feature under the rectangle. If there are many features under the rectangle, an arbitrary choice (from the user's view) is taken. If there is no feature completely under the rectangle, the connector's layer is blue (NMOS metal). After the Parm command is terminated, the data in the cell is clipped to the bounding box. The elements outside the box are left selected so

they can be easily deleted.

The clipping algorithm works as follows: if instances, boxes and polygons are not completely inside the bounding box of the cell, they are considered to be totally outside. Wires are clipped by chopping the center path so that the midpoint of the true end of the wire (not the end point) touches the edge of the bounding box. Irregular clipping occurs when a wire segment does not intersect the bounding box at a right angle. A segment touching the border at an arbitrary angle hangs one corner over the edge and leaves a wedge shaped gap inside the box. A corresponding angled segment in an adjacent cell will mesh properly. Wires that run near the border and parallel to it will be inside if they are more than half way inside.

When wires are broken at the bounding box, the two pieces have a common endpoint inside the bounding box. Therefore, if the clipping was not satisfactory, the pieces can be individually selected and Welded back together.

Editing Techniques

Because the LSI-11 is not infinitely clever about moving and writing over the Mark and the cursor, and because it is impossible to erase only the item deleted without erasing other features on the same layer, the display gets chewed up. If you are confused because something disappeared, try refreshing the screen by changing the magnification to the same value. The refreshed screen will accurately represent the data. To avoid cursor tracks, keep the cursor away from places where the screen is being modified.

The input points are taken off a grid which is the same as the tickmark grid. The only command which does not use the grid is the Select command, so it is possible to specify a rectangle that is truly inside a wire, without the corners jumping to points outside the rectangle.

Input Directly to the Database from Other Representations

Geom only has provisions in the user interface for creation of wires. It is possible,

however, to create boxes and polygons in the Geom data via internal calls. These calls could be done by other editors in the Desktop system or by the database-reading procedures. Features of boxes and polygons cannot be modified in Geom, but operations on boxes and polygons as a whole can be done. These are Delete, Copy, Move, Mirror and Rotate.

The client program should have GEODAT prefixing it. For details on interfacing to Geom, see <SSPLIB>GEODAT.SIM. The major classes and procedures of interest are:

wire -- A wire element.

```
REF(vector) path;
REAL width;
INTEGER layer;

REF(wire) PROCEDURE add(p); REF(point) p;
REF(wire) PROCEDURE remove(p); REF(point) p;
```

Path is the list of points that make up the path of the wire. Width is the width of the wire and layer is the layer of the wire. The layers are numbered as follows for NMOS:

1 - green - Diffusion
2 - red - Polysilicon
3 - blue - Metal
4 - black - Contact Cut
5 - yellow - Depletion Implant
6 - intense red - Buried Contact
7 - intense blue - Overglassing

The procedures add and remove the point, respectively. Add does not check to ensure the angles are multiples of 45 degrees. Remove checks for identical referents, (p == q) not equal values (p.equals(q)), so when removing a point, one must pass a pointer to the point to be removed.

Box -- A box with edges parallel to the coordinate axes.

```
REF(rectangle) gmbb;
INTEGER layer;
```

Gmbb is the rectangle which describes the box.

Polygon -- An unrestricted polygon.

```
REF(vector) path;
INTEGER layer;

REF(polygon) PROCEDURE add(p); REF(point) p;
REF(polygon) PROCEDURE remove(p); REF(point) p;
```

Path is the vector of points around the polygon.

Add and **remove** modify the wire path. Add does not check to ensure the angles are multiples of 45 degrees. Remove checks for identical referents, not p.equals(q), so when removing a point, one must pass a pointer to the point to be removed.

ginstance(cel) REF (basiccell) cel -- An instance of a cell.

```
REF(geomcell) gecel;
BOOLEAN visible;
REF(transform) trans;

REF(ginstance) PROCEDURE Xfrm(m); REF(transform) m;
```

Gecel is initialized when the ginstance is created. If none exists, one is created. **Xfrm** can be used to position the instance properly

connector -- A connector on a geomcell.

```
REF(connector) PROCEDURE named(t); [illegible] t;
REF(connector) PROCEDURE LocatedAt(r); REF(rectangle) r;
REF(connector) PROCEDURE OnLayer(l); INTEGER l;
```

The coordinates of the rectangle are in the defining cell's coordinate system. L is the layer of the connection.

geomcell -- A geometry cell data piece.

```
REF(vector) contents, connectors;

REF(geomcell) PROCEDURE wipe;
REF(geomcell) PROCEDURE add(e); REF elem(e);
REF(geomcell) PROCEDURE remove(e); REF elem(e);
REF(geomcell) PROCEDURE addconnector(c); REF connector(c);
```

Contents is the list of the data elements in the cell. A data element is a ginstance, box, polygon or wire. Connectors is the list of connectors on the cell.

Wipe clears all fields in geomcell in preparation for creation of new data. Add adds the specified element to the cell's data. Wires, boxes, polygons and ginstances are all subclasses of class elem. Therefore, all kinds of elements can be added to a cell with this procedure. Remove removes the specified element from the cell's contents list. Addconnector adds the connector to the cell's list of connectors.

How to Run GEOM

Geom can be found in <SSPLIB>GEOM.EXE. It needs no other files to run. If you are planning to write code in another Desktop representation to manipulate the Geom database, you will need, in addition to all the Desktop files, GREP.SIM, GEODAT.SIM and GEOM.SIM. GREP.SIM and GEOM.SIM can be left out if you do not intend to have a working Geometry editor in you Desktop system.